**Wsf Radar Sensor Analysis**

**文件路径**: src/radar/Sensor/WsfRadarSensor.cpp

**一、文件功能概述**

WsfRadarSensor.cpp 实现了 AFSIM 中雷达传感器的核心功能，包括:

1. **雷达模式管理**：加载和切换不同雷达工作模式(WsfSensorModeList)。
2. **发射与接收组件管理**：管理天线发射(WsfEM\_Xmtr)与接收(WsfEM\_Rcvr)组件。
3. **定时扫描与检测**：PerformScheduledDetections(double aSimTime) 方法负责周期性发射脉冲并调用 AttemptToDetect。
4. **探测尝试入口**：bool AttemptToDetect(double aSimTime, WsfPlatform\* aTargetPtr, Settings& aSettings, WsfSensorResult& aResult) 提供对单个目标的探测接口。
5. **结果通知**：完成检测后通过 NotifyTargetUpdated 和 NotifyTrackInitiated/Updated/Dropped 等方法分发检测结果。

**二、相关文件一览**

|  |  |
| --- | --- |
| **文件名** | **功能说明** |
| WsfAntennaPattern.cpp/hpp | 天线方向图与增益模型接口 |
| WsfEM\_Attenuation.cpp/hpp | 雷达信号传播衰减模型（对地表、多径、ITU 等） |
| WsfEM\_Clutter.cpp/hpp | 杂波模型（表面杂波、目标遮蔽等） |
| WsfEM\_Xmtr.cpp/hpp | 发射机组件，控制功率、频率、波形 |
| WsfEM\_Rcvr.cpp/hpp | 接收机组件，控制增益、噪声系数、滤波 |
| WsfRadarSensorErrorModel.cpp/hpp | 误差模型（系统漂移、校准误差） |
| WsfSensorSignalProcessor.cpp/hpp | 信号处理链（CTSR、FFT、检测门限） |
| WsfRadarSignature.cpp/hpp | 目标雷达散射截面(RCS)模型 |
| WsfTabularAttenuation.cpp/hpp | 表格化传播衰减插值模型 |

**注**: 以上文件归属于 src/radar/Sensor 和 src/radar/Engine 目录，用于分层实现传感器、传播与信号处理模型。

**三、关键方法解析**

**1. PerformScheduledDetections**

void WsfRadarSensor::PerformScheduledDetections(double aSimTime) {

// 1. 基于模式列表与调度器决定哪些目标需要检测

// 2. 遍历目标列表，调用 AttemptToDetect

// 3. 收集 WsfSensorResult 并分发通知

}

* **作用**：驱动整个雷达仿真的“心跳”，每次扫描都会执行此函数。
* **依赖**：mSchedulerPtr 决定扫描的时间和目标队列。

**2. AttemptToDetect**

bool WsfRadarSensor::AttemptToDetect(double aSimTime,

WsfPlatform\* aTargetPtr,

Settings& aSettings,

WsfSensorResult& aResult) {

// 1. 检查视场、距离与RCS

// 2. 传播衰减(WsfEM\_Attenuation)

// 3. 杂波抑制(WsfEM\_Clutter)

// 4. 信号处理(WsfSensorSignalProcessor)

// 5. 填充 aResult，返回是否检测

}

* **场景**：被 PerformScheduledDetections 在循环中多次调用。
* **输出**：aResult 填入检测概率(Pd)、目标距离、方位、径向速度等信息。

**3. 其他辅助模块调用**

* **传播模型**：WsfEM\_GroundWavePropagation、WsfEM\_ITU\_Attenuation 等，通过工厂模式选用。
* **方向图**：WsfAntennaPattern 提供天线增益与方向特性。
* **杂波模型**：不同 WsfEM\_ClutterTypes 用于定制化场地杂波评估。

**四、集成到 Qt UI 的建议实践**

1. **封装引擎接口**
   * 新建 class RadarEngine : public QObject，内部管理所有 WsfRadarSensor 及插件对象。
   * 定义信号：
   * signals:
   * void echoDetected(double azimuth, double range, double rcs);
   * void scanStarted();

void scanFinished();

1. **UI 触发与数据回调**
   * 在 MainWindow 中：

connect(startButton, &QPushButton::clicked, engine, &RadarEngine::startScanning);

* + RadarEngine::startScanning 调用 PerformScheduledDetections 并发射 scanStarted。
  + 将 echoDetected 连接到 PPI 控件，动态渲染回波点。

1. **插件管理**
   * 在 UI 中提供“插件列表”下拉框，动态加载不同传播/杂波/方向图插件。
   * 根据用户选择调用对应 WsfEM\_\* 插件接口。

**五、疑难排查要点**

* **插件加载顺序**：确保 SetModeList、SetScheduler、SetTracker 都在 Initialize 前完成。
* **线程安全**：若在后台线程跑仿真，需跨线程发射 Qt 信号。
* **实时性能**：可考虑先批量计算后再批量回调，避免 UI 频繁阻塞。

**六、缺失插件与扩展建议**

1. **场景管理与输入**
   * 当前缺少脚本解析引擎（WsfScriptContext）和场景配置插件，对仿真参数批量加载支持不足。
2. **平台/目标仿真**
   * WsfPlatform、WsfTrack 及相关动态平台模型源码未在此目录出现，需要额外集成 src/radar/Platform 和 src/radar/Track 子模块。
3. **调度器与跟踪器**
   * 默认 WsfDefaultSensorScheduler 与 WsfDefaultSensorTracker 实现需确认源码路径是否纳入，否则只能使用空调度/跟踪
4. **脚本化支持**
   * 如需图形化配置或后期扩展脚本化仿真，需要集成 script/WsfScriptContext.cpp/hpp 及相关脚本插件。

**建议**：将 src/radar/Platform、src/radar/Track 和 script 目录中的文件补充到项目中，确保形成完整的雷达仿真执行链路。

以上为完整分析与扩展建议。欢迎测试、补充并讨论！

**七、为什么要修改界面项目**

为了实现**用户界面与仿真引擎**的无缝协作，需要在界面项目中:

1. **触发仿真流程**
   * 将“开始仿真”按钮事件映射到仿真引擎核心方法(AttemptToDetect()、PerformScheduledDetections())
   * 确保用户操作能够驱动仿真引擎的执行
2. **解耦与维护**
   * 通过**信号/槽**机制，将仿真结果（回波数据、扫描状态）通知给 PPI 控件
   * 分离UI层与仿真层，增强代码可维护性和可扩展性
3. **实时反馈与响应**
   * 在仿真运行时，通过异步信号将检测结果推送给UI，保证PPI视图实时更新
   * 避免UI阻塞，提高响应速度和用户体验
4. **插件灵活管理**
   * 在界面中动态切换不同的传播/衰减/杂波插件，无需修改仿真引擎代码
   * 利于将来新增或替换不同模型

综上，**修改界面项目**是为了让UI能够正确调用仿真引擎、接收并展示结果，从而完成一个可交互、可扩展、实时响应的雷达显示系统。